

Mācību Centrs MP
Programēšanas Tehniks 2. Kurss

Prakses Atskaite Gameboy Emulātors

Autors: **Māris Muižnieks**

RĪGA 2022

Satura rādītājs

| | |
|--|----|
| 1. Ievads | 2 |
| 2. Organizācija un tajā izmantoto datorsistēmu apraksts | 3 |
| 3. Ergonomiskas darba vietas un darba vides izveidošana institūcijā..... | 4 |
| 3.1. Kas ir ergonomika? | 4 |
| 3.2. Iespējamie riski saistīti ar amatu | 4 |
| 4. Programmētāja profesijas standarts, galvenie uzdevumi, pienākumi un nepieciešamās prasmes | 5 |
| 5. Prakse..... | 5 |
| 5.1. Emulātors..... | 5 |
| 5.2. Gameboy | 5 |
| 5.3. Kartridžas | 6 |
| 5.3.1 Hедера čекsuma | 7 |
| 5.3.2 Kārtridžas ielāde | 8 |
| 5.4. Procesora emulācija | 9 |
| 5.4.1 Procesora struktūras | 9 |
| 5.4.2 Procesora funkcionalitāte | 11 |
| 5.5. Emulātora status..... | 12 |
| 6. Secinājumi un ierosinājumi | 13 |
| 7. Informācijas avoti | 14 |

1. Ievads

Praksēs mērķis veikt patstāvīgu darbu un pierādīt savas spējas strādāt patstāvīgi, pildīt uzdevumus, izmantot teorētiskās zināšanas praktiskā darbā un pilnveidot tās.

Prakses laikā strādāju SIA "Magnetic Professional" mācību centrs MP, reģ. Nr. 42103086895.

Prakses uzdevums bija izveidot GameBoy Emulātoru. Prakses laiks no 2022. gada 16. marta līdz 2022. gada 9. maijam. Prakses apjoms 240 Akadēmiskās stundas.

Programmēšanai tika izmantota C11 valoda, kā arī SDL2 bibliotēka. Emulātors tika veidots uz Mac OS Monterey 12.2 versija. Kā arī clang 13.0.0.

Šajā dokumentā aprakstīšu progamas izveidi, izmantotos informācijas resursus. Problēmas ar kurām sastapos, dažus no risinājumiem. Rīkus un sistēmas kuras izmantoju, kā arī secinājumus, piebildes un ietekumus.

2. Organizācija un tajā izmantoto datorsistēmu apraksts

Izmantoju Mac OS un Linux sistēmas kas ir bazētas uz unix tipa sistēmām.

Līdz ar to raksturošu kas ir Unix un kādēļ sistēmas bāzētas pēc Unix tipa sistēmas ir piemērotas Programmēšanai. Unix is daudzuzdevumu veikspējīga daudzlietotāju Operētāj Sistēma, kas tika radīta ATT Bell Labs 1969 gadā, to izveidotāji bija Kens Tompsons, Denis Riči un citi Bell Labs zinātnieki. Orģināli Unix bija paredzēta ka izdevīga platforma/OS prieks programētājiem dažādu programu izveidei. Ar laiku Sistēma agua un izpletās akadēmiskajā sfēra kur Lietotāji ar laiku izveidoja savus rīkus un sāka dalīties ar tiem. Sākotnēji Unix nebija veidots kā portabla vai Daudzuzdevumu veikspējīga Sistēma. Taču laika gaitā tās funkcionalitāte tika pilnveidota.

Unix Operētāj sistēma sastāv no daudzām bibliotēkām un programām ar galveno kontrol programu, tā saucamo "Kodolu", jeb Angliski "Kernel". Kodols rūpējas par dažādu programu darbības uzsākšanu un nobeigšanu, tas kontrolē failu sistēmas un citus bieži sastopamus "zemā līmeņa" darbus. Tas to dara lai programām būtu pieeja pie pašiem datora dzelžiem, failiem un apmainītos ar datiem starp citām programmām vienlaicīgi. Lai nodrošinātu šādu resursu menedžēšanu Kodolam pienākas speciālas priekšrocības, kas ir arī redzams Unix filozofijā par Kodola reģionu un Lietotāja reģionu kurā tad arī strādā lielākā daļa Lietotājprogrammas.

Mac OS ir Unix operētāj sistēma kas orģināli tika veidota no NextStep bāzes, kas savukārt tika veidota no BSD (Bearkley Software Distribution) bāzes un BSD jau šajā gadījumā nāca no paša Unix. Līdz ar to Mac OS tiek uzskatīta kā viena no īstajām Unix pēcteču operētāj sistēmām.

Savukārt Linux Sistēmu izveidoja Linus Torvalds, tā ir Atvērtā koda Operētāj sistēma kas tika veidota pēc Unix filozofijas. Mūsdienās tā ir viena no viss populārākajām OS priekš serveriem. Kā arī iegultajām sistēmām.

Koda rakstīšanai izmantoju EMACS jeb "Editor MACroS" teksta apstrādes programmatūru, tas ir plaši pazīstams teksta apstrādes rīks kas pazīstams ar savām paplašināšanas iespējām. Versija ko izmantoju ir GNU EMACS 28, kas tad ir arī viss populārākā versija.

Priekš kompilācijas izmantoju CLANG/LLVM kompilatoru. Pats Clang ir tikai priekšējā daļa pašam kompilātoram kas ir LLVM. LLVM ir diezgan liels projekts kas ir rakstīts C++ valodā un sevī ietver Kompilēšanas rīku komplektu. Clang šajā gadījumā ir LLVM priekšdaļa kas tülko C kodu uz LLVM atpazīstamu Objekt kodu. Ko tālāk jau LLVM pārtulko mašin kodā.

Priekš prakses atskaites sastādīšanas tika izmantota L^AT_EX teksta redakcijas valoda.

3. Ergonomiskas darba vietas un darba vides izveidošana institūcijā.

Lai darba vietā nodrošinātu ergonomisku darba vietu, tiek izmantoti Stāvgaldi, kas atļauj strādāt stāvus. Strādājot pie datora nepieciešams galdam atrasties augstumā lai, ar taisnu muguru, rokas elkoņu leņķis būtu 90°.

Sēdot vai stāvot pie galda, Datora monitoriem jābūt vienā augstumā ar acīm. Lai nodrošinātu acs kairināšanu pēc 45 min jā atiet no datora ekrāna, jāveic vingrinājumi.

3.1. Kas ir ergonomika?

Ergonomika ir darba procesa un darba vides piemērošana cilvēka psihiskajām un fiziskajām iespējām, lai nodrošinātu efektīvu darbu, kas neizraisa draudus cilvēka veselībai un kuru var viegli izpildīt

Ergonomiku var iedalīt 3 daļās:

- slodzes ergonomika, piemēram, fiziska slodze, piespiedu pozas, vienveidīgas kustības, smagumu nešana u.t.t.
- kognitīvā vai mentālā ergonomika, piemēram, mentālā slodze, u.t.t
- organizācijas ergonomika, piemēram, darba organizācija, procesi, atpūta, u.t.t

3.2. Iespējamie riski saistīti ar amatu

Iespējamie riski saistīti ar programmēšanas amatu ir:

- Darbs piespiedu pozā
- Vienveidīgas kustības
- Mentālā slodze

4. Programmētāja profesijas standarts, galvenie uzdevumi, pienākumi un nepieciešamās prasmes

Profesijas standarts ir ar profesijas kodu 2512 05, tas ir ar ceturtās profesionālās kvalifikācijas līmeni.

Īsumā Programētājam jāprot izstrādāt programmatūra atbilstoši funkcionalitātes, kvalitātes un resursietilbības nosacījumiem. Jāprot konfigurēt izstrādes vidi un raksturot programmas kodu saskaņā ar projektējumu un kodēšanas vadlīnijām. Jāprot veikt vides sagatavošanu programatūras ieviešanai, jāievieš un jāuztur programatūru un jāpiedalās programatūras projektu plānošanā.

5. Prakse

Prakses mērķis bija izveidot GameBoy emulātoru. Turpmākajās sadaļās ieskicēšu kas īsti ir emulātors, kas ir pats GameBoy, kā to veidoju, kādus risinājumus izmantoju.

5.1. Emulātors

Datorzinātnēs emulātors ir programma kas ļauj sistēmai imitēt/simulēt kādas citas sistēmas darbību. Piemēram liela daļa printeru emulē HP LaserJet printerus, jo liela daļa programmatūras ir izveidota tieši priekš HP printeriem. Tādējādi printeris kas nav HP ražots spēj izmantot programmatūras kas ir domātas priekš šiem HP printeriem. Vai arī programa DOSBox kas emulē DOS operētāj sistēmu un ļauj izmantot programmatūru kas tika veidota priekš DOS sistēmas.

5.2. Gameboy

Gameboy ir 8bitu pārnēsājamā spēļu konsole ko izveidoja un ražoja Japāņu uzņēmums Nintendo. To izlaida Japānā 1989 gada 21. aprīlī, un tā paša gada 31 Jūlijā ASV un Eiropā 1990 gada 28. Septembrī. Gameboy priekš sava laika bija avancēta sistēma kura izmantoja speciālu 8bitu Sharp Mikroprocesoru (Sharp LR35902) kas strādāja ar 4.19MHz takts frekvenci. Kam ir 64KB adresēšanas lauks ko sastāda šādi:

- 8KB iebūvētais RAM
- Līdz 16 8KB ārējais maināms RAM (kas ir uz kārtīdžas) maksimāli līdz 128KB kopējais ārējais RAM. (parasti tika izmantots Spēļu saglabāšanai)
- 8KB RAM priekš grafikas apstrādes un displeja

Gameboy strādā ar 2bitu krāsām, efektīvi 4 pelēkiem toņiem. Tā LCD Displeja izšķirtspēja ir 160x144 pixeli. Tam ir 8 ievad pogas: 4 virzienu pogas, start, select, A un B.

5.3. Kartridžas

Nedaudz par pašām Kārtridžām, par cik emulācija notiek programmā, tad reālas kārtridžas netiek izmantotas, taču tiek izmantoti tā saucamie ROM faili. Kas nozīmē ka Viens no pirmajiem uzdevumiem ir ielādēt programmā šo te ROM faila saturu. Tas nozīmē ka vajag saprast kas par informāciju mums ir nepieciešama un kas tā ir.

Tālāk ar "0xFFFF" tiks apzīmēti Heksadecimālie skaitļi, ar kuriem apzīmējama atmiņas adrese.

No 0x0100 līdz 0x014F ir ROM faila, jeb kartridžas informācijas abgabals:

- 0x0100-0x0103 Sākum punkts, šis ir punkts, uz kuru parasti pēc Nintendo Logo parādīšanas konsolē, procesors lec uz šo adreses reģionu kur parasti ir sastopamas NOP un jp 0x0150 instrukcijas kas tālāk jau uzsāk pašas programmas/spēles darbību.
- 0x0104-0x0133 satur jau pieminēto Nintendo logo, BITMAP formātā, kas tiek izmantots anti-pirātiskās čeksummas pārbaudei.
- 0x0134-0x0143 Spēles nosaukums.
- 0x013F-0x0142 Ražotāja kods
- 0x0143 GameBoy Color karogs. atbild par Color funkcionalitātes atbalstu.
- 0x0144-0x0145 Licences kods Norāda kāds izdevējs ir šai spēlei.
- 0x0146 Super Gameboy karogs, atbild par Super Gameboy atbalstu.
- 0x0147 Kartridžas tips.
- 0x0148 ROM izmērs.
- 0x0149 RAM izmērs, ja tāds ir.
- 0x014A Reģiona specifikācija, vai spēle ir priekš Japānas tirgus vai nē.
- 0x014D Hedera čeksuma.
- 0x014E-0x014F Satur galveno, globālo čeksumu, pats Gameboy šo nepārbauda.

5.3.1. Hedera čeksma

Tātad viena no galvenajām lietām, pēc Kartridžas/Rom datu ielādes būtu pārbaudīt šo te Hedera čeksma. To varam pārbaudīt ar šo kodu:

```
1 u16 x = 0;
2 for (u16 i = 0x0134; i <= 0x014c; i++)
3     x = x - ctx.rom_data[i] - 1;
```

Šajā gadījumā `ctx.rom_data[]` ir struktūras masīvs kurā esam ielādējuši ROM faila datus. Pēc cikla izpildes varam salīdzināt mainīgo `x` ar pašu čeksma, jeb pašas čeksma apakšējajiem 8 bitiem. Ja šajā gadījumā čeksma nesakrīt tad Kartridžu nevar lādēt.

5.3.2. Kārtridžas ielāde

Pašus kārtridžas datus ielādējam nodefinētā konteksta struktūrā.

```
1 typedef struct {
2     char fn[1024];           // Faila nosaukums
3     u32 rom_size;           // ROM faila izmērs
4     u8 *rom_data;           // Paši ROM faila dati
5     rom_header *header;     // Hedera informācija
6 } cart_context;
```

Struktūra sastāv no 4 elementiem, no kuriem viens ir papildus struktūra ar Hedera informāciju.

```
1 typedef struct {
2     u8 entry[4];             // Programmas sākums
3     u8 logo[0x30];          // NINTENDO loggo
4
5     char title[16];          // Nosaukums
6     u16 new_lic_code;        // Licences kods
7     u8 sgb_flag;            // Super Gameboy Karogs
8     u8 type;                 // Tips
9     u8 rom_size;            // ROM izmērs
10    u8 ram_size;             // RAM izmērs
11    u8 dest_code;            // Reģiona kods
12    u8 lic_code;             // Vecais licences kods
13    u8 version;              // Versija
14    u8 checksum;             // Čeksumma 0x014D
15    u16 global_checksum;     // Globā čeksumma
16 } rom_header;
```

Kad šīs struktūras tiek aizpildītas ar informāciju varam doties tālāk uz pašu emulāciju un domāt par to kā strādās mūsu emulētais processors.

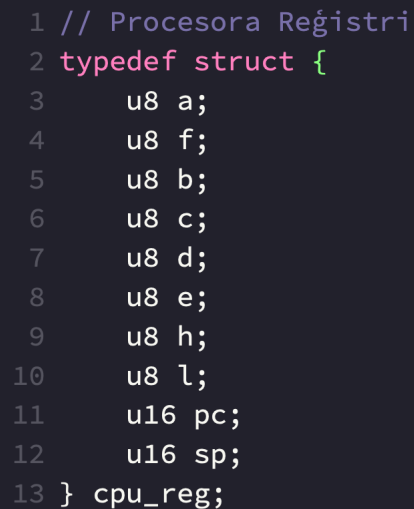
5.4. Procesora emulācija

Lai efektīvi emulētu procesoru, tas ir principā jāizveido Programmā no jauna. Tas nozīmē ka tam ir jāpilda instrukcijas, jāizmanto reģistri, karogi, adresēšanas veidi u.t.t.

5.4.1. Procesora struktūras

Priekš tā mums ir nepieciešams izveidot datu struktūras kas saturētu Procesora kontekstu dotā mirklī.

Priekš tā mums ir pati galvenā procesora struktūra.



```
1 // Procesora Reģistri
2 typedef struct {
3     u8 a;
4     u8 f;
5     u8 b;
6     u8 c;
7     u8 d;
8     u8 e;
9     u8 h;
10    u8 l;
11    u16 pc;
12    u16 sp;
13 } cpu_reg;
```

Sākam ar procesora reģistriem, realitātē Gameboy procesoram ir 16bitu reģistri, un tie ir tikai 6 (AF, BC, DE, HL, SP, PC) taču par cik pirmie 4 ir dubult reģistri, tad veidojam tos atsevišķi lai atvieglotu savu dzīvi. SP jeb steka rādītājs un PC jeb Programmas skaitītājs paliek kā 16bitu skaitļi jo šie divi reģistri tieši ir atbildīgi par procesora adresēšanu. Vel viens svarīgs reģistrs ir F, kas ir karogu reģistrs, tas sevī satur 4 karogus, Z, N, H, C. Kur Z ir Nulles karogs, S ir atņemšanas karogs, H ir Pus pārnese karogs, un C ir pārnese karogs. Tas nozīmē ka mums būs nepieciešams ar Bitu līmeņa operācijām, lai piekļūtu tiem.

```

1 typedef struct {
2     cpu_reg reg;           // Procesora reģistri
3     u16 fetched_data;      // Iegūtie dati
4     u16 mem_dest;         // Atmiņas galamērķis
5     bool dest_is_mem;     // Vai galamērķis ir Atmiņa
6     u8 cur_opcode;        // Patreizējais Instrukcijas Kods
7     instruction *cur_inst; // Patreizējā instrukcija
8
9     bool halted;          // Apturēts
10    bool stepping;
11    bool int_master_enabled; // Apturēšanas ieslēgta
12    bool ime_enable;       // Ieslēgt apturēšanu
13
14    u8 ie_register;        // Apturēšanas Reģistrs
15    u8 inter_flag;        // Apturēšanas karogs
16
17 } cpu_context;

```

Šeit ir arī paša procesora konteksta struktūra. Kas satur sevī reģistrus, Apturēšanas funkcionalitātes, instrukcijas u.t.t.

Instrukciju struktūra sastāv no:

```

1 typedef struct {
2     in_type type;          // Instrukcijas tips
3     addr_mode mode;       // Adresēšanas veids
4     reg_type reg_1;       // reģistra tips
5     reg_type reg_2;       // reģistra tips
6     cond_type cond;       // Apstākļa kods. Karogi
7     u8 param;            // parametrs, parasti adrese
8 } instruction;

```

Kā redzams, šīs struktūras ir arī lielās problēmu radītājas pie emulācijas, jo struktūras veidojas ļoti sarežģītas un tās kaut, kā vajag arī dabūt līdz funkcijām.

5.4.2. Procesora funkcionalitāte

Emulātors šajā gadījumā iet soli pa solim, pēc Steka Rādītāju un Programas skaitītāju, un tad jau ņem instrukcijas, tās dekodē, atrod pareizo instrukciju pēc konteksta, un to izpilda.

```
1 static void cpu_execute()
2 {
3     IN_PROC proc = inst_get_processor(ctx.cur_inst->type);
4
5     if (!proc) {
6         printf("err... Fail\n");
7         exit(EXIT_FAILURE);
8     }
9     proc(&ctx);
10 }
```

Šeit izveidojam, funkcijas rādītāju, ja neizdodās izveidot rodas programmas kļūda. Tam pēctam padodam kontekstu.

```
1 // Funkciju rādītāju masīvs
2 static IN_PROC processors[] = {
3     [IN_NONE] = proc_none,
4     [IN_NOP] = proc_nop,
5     [IN_LD] = proc_ld,
6     [IN_LDH] = proc_ldh,
7     [IN_JP] = proc_jp,
8     [IN_DI] = proc_di,
9     [IN_POP] = proc_pop,
10    [IN_PUSH] = proc_push,
11    [IN_JR] = proc_jr,
12    [IN_CALL] = proc_call,
13    [IN_RET] = proc_ret,
14    [IN_RST] = proc_rst,
15    [IN_DEC] = proc_dec,
16    [IN_INC] = proc_inc,
17    [IN_ADD] = proc_add,
18    [IN_ADC] = proc_adc,
19    [IN_SUB] = proc_sub,
20    [IN_SBC] = proc_sbc,
21    [IN_AND] = proc_and,
22    [IN_XOR] = proc_xor,
23    [IN_OR] = proc_or,
24    [IN_CP] = proc_cp,
25    [IN_CB] = proc_cb,
26    [IN_RRCA] = proc_rrca,
27    [IN_RLCA] = proc_rlca,
28    [IN_RRA] = proc_rra,
29    [IN_RLA] = proc_rla,
30    [IN_STOP] = proc_stop,
31    [IN_HALT] = proc_halt,
32    [IN_DAA] = proc_daa,
33    [IN_CPL] = proc_cpl,
34    [IN_SCF] = proc_scf,
35    [IN_CCF] = proc_ccf,
36    [IN_EI] = proc_ei,
37    [IN_RETI] = proc_reti
38 };
39
40 // Funkcija kas izmanto masīvu lai atrastu funkciju pēc tā tipa
41 IN_PROC inst_get_processor(in_type type)
42 {
43     return processors[type];
44 }
```

Šeit var redzēt kā pēc instrukciju tipa tiek izsaukta pareizā funkcija. Kas tālāk jau veic darbības pēc paplašināta konteksta un informācijas par instrukciju.

```

1 // AND instrukcija
2 static void proc_and(cpu_context *ctx)
3 {
4     ctx->reg.a &= ctx->fetched_data;
5     cpu_set_flags(ctx, ctx->reg.a == 0, 0, 1, 0);
6 }

```

Šeit redzams kā tiek veikta and operācija Vicam bitu līmeņa AND operāciju ko saglabājam tur pat A reģistrā, ieslēdzam nepieciešamos karogus un ar to and operācija ir izpildīta.

5.5. Emulātoru status

Kopumā emulātoru izveide ir sarežģīts uzdevums. Uz doto mirkli Emulātors strādā ar daudz kļūdām un tam nav skaņas. taču tas spēj ielādēt ROM failu, un spēj palaist to. Taču rodas artefakti grafiskajā izdevē un daži ROM faili, aptur Emulātoru darbību.



Šeit redzama spēle TETRIS ko izmantoju emulātoru testēšanai. Titula ekrāns strādā un izvada ekrānā attēlu bez redzamiem artefaktiem. Taču kad mēģina spēlēt spēli veidojas artefakti, līdz ar to redzams ka emulātors nav līdz galam pabeigts un ir kļūdainš.



6. Secinājumi un ierosinājumi

Secinājumi pēc prakses ir vienkārši. Izvēlētais prakses uzdevums iespējams ir par sarežģītu, taču rezultātā esmu uzzinājis arī daudz ko jauni. Uzskatu ka prakse ir aizvadīta veiksmīgi, lai gan tā ir bijusi grūta taču nevarētu teikt, ka tā nav izveusies.

Prakses laikā izdevās izveidot pamata emulācijas programmu priekš Gameboy sistēmas, tā spēj ielādēt ROM failus un spēj veikt pamata Gameboy funkcijas, taču tā nav līdz galam pabeigta, un mēģinot spēlēt, spēles caur emulātoru veidojās dažādi vizuālie artifakti, vai emulātorā programma, vienkārši apstāj savu darbību.

Ieteikumi, būtu veidot garāku prakses laiku, it īpaši strādājošajiem studentiem, kam prakses apvienošana ar darbu sagādā sarežģītības.

7. Informācijas avoti

Gameboy Izstrādes dokuments.

Informācija par Gameboy

Informācija par Gameboy instrukcijām

Informācija par GameBoy Emulāciju

Informācija par Emulātoriem

Informācija par Gameboy Wikipedia

Informācija par UNIX

Informācija par Linux

Informācija par Emacs

informācija par Ergonomiku